



TECHNICAL REPORT

**SmartM2M;  
oneM2M Performances Evaluation Tool (Proof of Concept)**

---

**Reference**

DTR/SmartM2M-103841

---

**Keywords**evaluation, KPI, oneM2M, performance, scenarios,  
simulation**ETSI**650 Route des Lucioles  
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B  
Association à but non lucratif enregistrée à la  
Sous-Préfecture de Grasse (06) N° w061004871

---

**Important notice**

The present document can be downloaded from:

<https://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status.

Information on the current status of this and other ETSI documents is available at

<https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:

<https://portal.etsi.org/People/CommitteeSupportStaff.aspx>

If you find a security vulnerability in the present document, please report it through our  
Coordinated Vulnerability Disclosure Program:

<https://www.etsi.org/standards/coordinated-vulnerability-disclosure>

---

**Notice of disclaimer & limitation of liability**

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

---

**Copyright Notification**

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2024.  
All rights reserved.

# Contents

Intellectual Property Rights .....	4
Foreword.....	4
Modal verbs terminology.....	4
Introduction .....	4
1 Scope .....	5
1.1 Context for the present document.....	5
1.2 Scope of the present document.....	6
2 References .....	6
2.1 Normative references .....	6
2.2 Informative references.....	6
3 Definition of terms, symbols and abbreviations.....	7
3.1 Terms.....	7
3.2 Symbols.....	7
3.3 Abbreviations .....	7
4 oneM2M profiler .....	8
4.0 Introduction .....	8
4.1 Structure .....	8
4.2 Operating system interaction.....	10
4.3 Profiler usage.....	10
4.3.1 Installation and configuration .....	10
4.3.2 Execution .....	10
4.4 Profiler output format.....	11
5 oneM2M IoT system simulator .....	11
5.0 Introduction .....	11
5.1 OMNeT++.....	11
5.2 oneM2M deployment model .....	12
5.3 Mapping on OMNeT++ library and the NED topology .....	16
5.3.1 Mapping of Meta Model on OMNeT++ .....	16
5.3.2 Mapping of the probes in the simulator .....	17
5.4 Installation and configuration.....	17
6 Frameworks at work.....	18
6.1 Mapping Scenario Description .....	18
6.1.1 Running example: A traffic light system .....	18
6.1.2 OMNeT++ files .....	19
6.1.3 Synthetizing KPIs .....	21
6.2 Performance evaluation analysis .....	21
7 Conclusions .....	24
<b>Annex A: Source code.....</b>	<b>25</b>
History .....	26

---

# Intellectual Property Rights

## Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: "*Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards*", which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<https://ipr.etsi.org/>).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

## Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

---

# Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Smart Machine-to-Machine communications (SmartM2M).

---

# Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](#) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

---

# Introduction

ETSI TTF T019 aims at studying, analysing, evaluating and simulating IoT application deployments on some oneM2M open-source implementations. Evaluation will be conducted based on case studies, deployment model and performance KPIs (Key Performance Index), all described in ETSI TR 103 839 [i.1] and ETSI TS 103 840 [i.2].

The objective of the present document is to describe and present the simulation tool and the profiler tool developed to that aim. The simulation tool is a OMNeT++ library implementing the deployment scenario models, and the case studies as described in ETSI TR 103 839 [i.1] and ETSI TS 103 840 [i.2]. The profiler is a standalone software to be run together with a real open-source oneM2M implementation dealing from scripts creating oneM2M "burst" till real case study. The profiler provides KPIs values concerning the oneM2M implementation performance and/or the case study: these KPI values will be used as parameters in the OMNeT++ simulator allowing a realistic large-scale simulation.

---

# 1 Scope

## 1.1 Context for the present document

The oneM2M standards are now mature: multiple deployments exist all over the world at both experimental and operational levels. The experimental deployments are conducted for multiple reasons:

- to evaluate the capabilities of the standard in terms of expressiveness, usability on specific equipment, connection with specific existing systems or performance evaluation;
- to provide a methodological study, based on performance evaluation (time, space) on a given set of "paradigmatic use cases";
- to measure KPIs defined in the present document of implementations that are compliant with the oneM2M standard, available either freely or commercially.

Use cases are evaluated in terms of chosen KPI: e.g. running time, memory space, numerosity of oneM2M entities (e.g. AE, MN-CSE, CSE), data transfer volume and real-time needs. Using a selected set of available oneM2M CSE implementations [i.9], a simulation library or an ad hoc simulator is to be provided, offering the ability to evaluate and simulate the performance of the use cases and give crucial information/feedback to the general user of oneM2M to choose and tune their IoT applications based on oneM2M framework [i.5]. The results of this tool development and evaluations of the use cases will be the basis to generate other deliverables. The present document was developed in the context of ETSI TTF T019, set up to perform work on "Performance Evaluation and Analysis for oneM2M Planning and Deployment". Five elements were addressed sequentially:

- 1) A collection of **use cases and derived requirements** were formally identified and defined. This work includes identification of relevant deployment scenarios. The present document adopted the use case style and template from oneM2M with a minor modification to address some performances issues. This phase of the work resulted in deliverable ETSI TR 103 839 [i.1].
- 2) The definition of **performance evaluation model**, with specification of procedures to assess the performance of oneM2M-based IoT platforms. This includes the identification and definition of a set/list of KPIs necessary to assess the deployment. For those KPIs, provision of a formal description of the test campaign and the test results to be obtained. This phase of the work resulted in deliverable ETSI TS 103 840 [i.2].
- 3) The creation of a **proof of concept** of a performance evaluation tool. This work also relies on a formal description of the identified deployment scenarios (single vertical domain & multiple vertical domains). This phase of the work resulted in the present document.
- 4) A practical **demonstration and analysis** exercise putting the proposed tool to use, with a specific oneM2M implementation but aimed at being a blueprint for the adoption and re-use of the results of ETSI TR 103 839 [i.1], ETSI TS 103 840 [i.2], and the present document with other oneM2M implementations and deployment scenarios. This phase of the work will be used in deliverables ETSI TR 103 842 [i.3] and ETSI TR 103 843 [i.4].
- 5) The development of a set of **guidelines and best practices** documenting best practices and lessons learnt as well as providing instructions for IoT solution topology, capacity provisioning, and expected performances that will give crucial directives and information to designer and implementors. This phase of the work resulted in deliverable provisioning and expected performances ETSI TR 103 843 [i.4].

The present document covers the third of the five items listed above and provides the basis for the related ETSI publications listed below:

- ETSI TR 103 841 (the present document).
- ETSI TR 103 842 [i.3].
- ETSI TR 103 843 [i.4].

## 1.2 Scope of the present document.

The present document presents the tools and libraries developed. More precisely:

- 1) A **Profiler tool**, written in Python, whose purpose is to listen or generate requests to a oneM2M CSE implementation, capture the responses and build a trace file in a given *ad hoc* format.
- 2) An **OMNeT++ simulation library**, written in C++, coupled with the NED proprietary OMNeT++ language, used to specify the case studies graph topologies, as described in [i.10].

The present document is structured as follows:

- Clauses 1 to 3 provide background and references including the definition of terms, symbols and abbreviations, which are used in the present document.
- Clause 4 describes the profiler tool that will be run along with an open-source oneM2M implementation, its internal structure, including its interactions with operating system (e.g. Linux®).

NOTE: Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

- Clause 5 describes the discrete event OMNeT++ simulator and its instantiation of a deployment model applied on oneM2M case studies, as described in [i.1] and [i.2]. This includes the OMNeT++ library and the NED topology specifications and oneM2M protocol interconnection rules.
- Clause 6 puts at work the above elements to the case studies defined in ETSI TR 103 839 [i.1], highlighting the simulation capabilities of the library, according to the KPI synthesized by the profiler.
- Clause 7 provides the conclusions of this work and present some potential improvements and extensions.

---

## 2 References

### 2.1 Normative references

Normative references are not applicable in the present document.

### 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document, but they assist the user regarding a particular subject area.

- |       |   |
|-------|---|
| [i.1] | ETSI TR 103 839: "SmartM2M; Scenarios for evaluation of oneM2M deployments".  |
| [i.2] | ETSI TS 103 840: "SmartM2M; Model for oneM2M Performance Evaluation".   |
| [i.3] | ETSI TR 103 842: "SmartM2M; Demonstration of Performance Evaluation and Analysis for oneM2M Planning and Deployment". |
| [i.4] | ETSI TR 103 843: "SmartM2M; oneM2M deployment guidelines and best practices".   |
| [i.5] | <a href="#">oneM2M TS-0001 (V3.34.0)</a> : "Functional Architecture".   |
| [i.6] | <a href="#">oneM2M TS-0008 (V3.9.0)</a> : "CoAP Protocol Binding".  |
| [i.7] | <a href="#">oneM2M TS-0009 (V3.9.0)</a> : "HTTP Protocol Binding".  |

- [i.8] [ETSI TS 118 110 \(V3.1.0\)](#): "oneM2M; MQTT Protocol Binding (oneM2M TS-0010 version 3.1.0 Release 3)".
- [i.9] oneM2M - [List of oneM2M deployments](#).
- [i.10] OMNeT++: "[Discrete Event Simulator](#)".
- [i.11] [Tools Deliverable ETSI repository](#) linked with the present document.
- [i.12] OMNeT++: "[Installation Guide](#)".

## 3 Definition of terms, symbols and abbreviations

### 3.1 Terms

For the purposes of the present document, the following terms apply:

**guidelines and good practices:** methodological document that gives hints to deploy a oneM2M infrastructure

**Key Performance Index (KPI):** list of criteria to be measured on a system

**oneM2M deployment:** mapping of a IoT applications on a oneM2M infrastructure

**performance evaluation:** evaluation of temporal, data transfer volumetry, and scalability aspects of a system

**platform evaluation tool:** simulation environment that is used to calculate/demonstrate the performance of a system

**profiler:** monitoring tool measuring KPIs

**real time constraints:** dynamic constraints to be fulfilled related to time

**single/multiple horizontal/vertical domains:** interaction capability of many oneM2M infrastructures from different domains

### 3.2 Symbols

Void.

### 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ADN	Application Dedicated Node
AE	Application Entity
CIN	Content INstance
CNT	CoNTainer
CoAP	Constrained Application Protocol
CPU	Central Process Unit
CRUD	Create, Read, Update, Delete
CSE	Common Service Entity
ETSI	European Telecommunications Standards Institute
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IN-CSE	Infrastructure Node - Common Services Entity
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Index
M2M	Machine-to-Machine
Mca	Reference Point for M2M Communication with AE
Mcc	Reference Point for M2M Communication with CSE

MIPS	Million Instructions Per Second
MN-CSE	Middle Node - Common Services Entity
MQTT	Message Queue Telemetry Transport
NED	NEtwork Descriptor
OASD	oneM2M Application Scenario Descriptor
OCPD	oneM2M CSE Performance Descriptor
OM2M	Eclipse OM2M - Open-Source platform for M2M communication
OSDD	oneM2M Solution Deployment Descriptor
PER	Packet Error Rate
SDK	Software Development Kit
TCP	Transport Control Protocol
TR	Technical Report
TS	Technical Specification

---

## 4 oneM2M profiler

### 4.0 Introduction

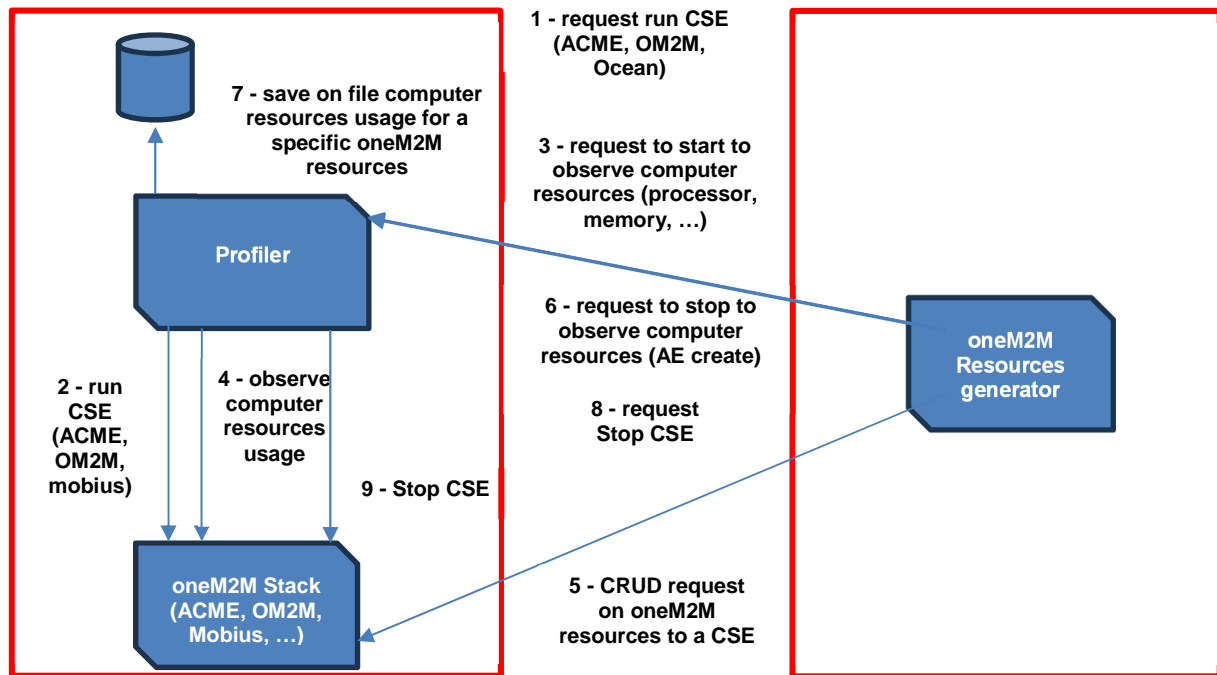
The simulation of an IoT infrastructure in terms of equipment, service layer and application require calibration of the inputs of the simulator. The different layers of the model proposed in ETSI TS 103 840 [i.2] have an impact on the overall system. The aim here is to propose a measurement architecture making it possible to extract measurements from real execution on given equipment, from a specific oneM2M implementation for a particular operation. Those measurements should be generalized at the simulator level to give KPIs having values as close as possible to the reality of deployment on other equipment for IoT applications with different configuration and behaviour.

### 4.1 Structure

The minimum oneM2M-level configuration needed to perform measurements requires a CSE and a client application making REST requests to the CSE. These two entities can be on the same equipment or on different equipment communicating via a network. Added to this is an entity called profiler responsible for observing the CSE instance and its use of the resources of the hardware equipment. These observations are then processed by the profiler and saved on permanent media in a file.

The implementation of a CSE involves complex mechanisms such as the use of thread, processing priority, recovery, cache, etc. The observation of each of these mechanisms and their impact can only be done by direct insertion of a probe into the code of a CSE itself. This makes the approach not portable and restricted to experts in a specific stack. The choice for profiling was to treat the CSE as a black box and to put probes at the operating system level and therefore the resource requests made by the CSE to the operating system of a machine. This approach makes the profiler independent of CSE implementations but still maintains a dependence on the type of operating system. The precision of the metrics obtained remains sufficient to detect and give the order of magnitude of saturation phenomena (for example of memory) or inadequate response time for a given use case.





**Figure 4.1-1: Interactions between the profiler and the oneM2M system**

Figure 4.1-1 gives an overall view of the system and the exchanges with the different steps of the profiling. In the initial state, the profiler is executed on a machine and an oneM2M query generation client is launched, then the experimental protocol is started:

- 1) The query generator begins by asking the profiler to launch and start monitoring a particular oneM2M CSE.
- 2) The CSE is executed on the machine.
- 3) The query generator warns the profiler that has to begin its observation on a particular sequence of queries that it will perform on the CSE.
- 4) The profiler launches its observation.
- 5) The query generator executes a set of queries to the CSE. This sends the oneM2M primitives that arrive on the Mca or Mcc interfaces of the CE implementation.
- 6) The query generator notifies the profiler that the queries on the CSE have been completed and that it can generate the observation statistics of the hardware resources used by the CSE.
- 7) The profiler saves its observations in a file which will be used as input to the simulator.
- 8) When the entire profiling plan is completed, the query generator asks the profiler to stop the CSE.
- 9) The profiler stops the CSE and terminates.
- 10) The query generator ends.

This simple exchange protocol allows to configure various query generation scenarios. To do this, it is enough to define the oneM2M query plan at the query generator level and on the other hand to explain the method for calculating the statistics on the profiler side for this particular query sequence.

## 4.2 Operating system interaction

In this "black box" profiling model, there is a strong dependency on the operating system. Those provide libraries for relating the use of hardware and system resources by a particular running process. There are many possible metrics. Here too, the structure of the profiler as an independent entity on the same machine as the CSE makes it possible to enrich the metrics according to needs. For example, in the UNIX world and more particularly Linux, all the files under the /proc directory allow to know the use of resources via measurements directly in the kernel or via counters. For example, the "stat" file gives information on the number of processor ticks used by the process, the "statm" file gives information on the use of RAM and virtual memory, etc.

## 4.3 Profiler usage

### 4.3.1 Installation and configuration

The installation and configuration process involves the oneM2M implementation to observe, the profiler and the query generator.

**Step 1:** Install a oneM2M implementation to observe on the target deployment machine. The oneM2M CSE implementation needs to be configured according to the instructions of that implementation; i.e.name of the CSE, address, etc. It is noted that version 1 of query generator only accepts the http protocol.

**Step 2:** Install the profiler and the query generator from the ETSI git repository. These programs are written in Python and require the Python interpreter to be installed on the CSE machine and the query generator machine.

**Step 3:** Configure the profiler. The profiler program must be put in the directory where the CSE code is located. In version 1 of the profiler only CSEs from the ACME, Mobius and OM2M distributions are created. Three parameters must be configured in the profiler:

- 1) the HOST variable specifying the IP address of the profiler;
- 2) the PORT specifying the port which will be used to communicate with the profiler;
- 3) the MIPS (million instructions per second) constant for the processor processing capacity in term of MIPS.

**Step 4:** Configure the query generator. The connection with the profiler must be made through the HOST and PORT variables of the generator which has the same values as the HOST and PORT variables of the profiler. Then it is needed to configure the information concerning the oneM2M implementation through the variables CSE\_URL\_XXX and ORIGIN\_XXX (XXX is to be replaced by the correct oneM2M implementation name) which gives respectively the URL of the CSE and the authentication necessary for connection.

**Step 5:** Configure the connection with an http server if the use case includes the oneM2M notification mechanism via the HTTP\_SERVEUR variable. For example, the ACME notification server can be used.

### 4.3.2 Execution

Once the system is configured as described in clause 4.3.1. the simulation can be executed.

**Step 1:** Run the profiler via the command: "python3 profiler.py".

**Step 2:** Run the query generator. Two modes exist:

- 1) Interactive mode: "python3 oneM2M\_Ressources.py manual", in this mode an interaction with the user allows the user to choose the IoT stack, the type of resource, the number of operations to be done on the same type of resources to generate averages.
- 2) Automatic mode: "python3 oneM2M\_Resources.py auto [OM2M/ACME/MOBIUS] [number of iterations per test]", in this mode a pre-defined scenario allows the user to create and destroy the most classic oneM2M resources by choosing the stack to execute and the number of requests per resource to generate the statistic.

These two executions generate a file with the statistics indexed by the name of the stack. The output file named STACKNAME\_resources is in the directory where the profiler tool is located.

## 4.4 Profiler output format

The profiler generates a text file. The format of this text file is:

- a first line gives an information on performance of the machine with indication of MIPS;
- all other lines have the format: type of oneM2M resources, type of action and a list of hardware type and value measured.

The first list of hardware resource on this first version is:

- cpu: average time of processors usage in second;
- mem: average size of memory in bytes.

EXAMPLE:     *MIPS 4589*  
           *AE, Create, cpu, 0.023333333333333428, mem, 43690.6666666666664*  
           *AE, Delete, cpu, 0.013333333333333345, mem, 43690.6666666666664*  
           *CNT, Create, cpu, 0.020000000000000018, mem, 43690.6666666666664*  
           *SUB, Create, cpu, 0.03333333333333336, mem, 87381.333333333333*

# 5 oneM2M IoT system simulator

## 5.0 Introduction

This clause presents the oneM2M IoT system simulator and its associated results in terms of performance evaluation. The objective is to present and explain the main phases of the instantiation, in the OMNeT++ environment, of a use case proposed in ETSI TS 103 840 [i.2] and its associated oneM2M multi-layered model proposed into the present document. The resulting simulator takes as inputs the KPIs produced by the oneM2M profiler presented in clause 4. The simulator produces itself performance evaluation results of the system also called KPIs.

### 5.1 OMNeT++

OMNeT++ is an extensible, component-based C++ simulation library and framework (Integrated Development Environment (IDE)), dedicated to distributed/networked systems simulations. The OMNeT++ environment is open source and thus, federates a large community who develops different features and libraries for specifying, editing, programming, and simulating networked systems.

The simulation engine is a discrete event simulator that can handle multiple nodes and their network topology. Depending on the fine or coarse grain modelling-level of the system under evaluation, the environment can provide either predefined libraries that implements full protocol stacks such as TCP/IP, MQTT, or enables a more or less accurate programming of the communication protocols and nodes behaviour, their interconnections, as well as the performance parameters to be measured. The integrated discrete event simulation engine also supports parallel distributed simulation to run large-scale topologies and system's behaviour.

The programming features of the IDE are intensively used for the development of a oneM2M system simulator. The notions of *modules* (simple or compound), *gates*, *channels*, *network*, *interfaces* are the inner elements of an OMNeT++ specification. Each element is associated with a C++ class with the associated properties, attributes, variables, and behaviour. The element behaviour is developed in two C++ methods namely: *initialize()* and *handleMessage()*. The method *initialize()* is called at the initial stage (transitional phase) where OMNeT++ elements of these different types are instantiated in the simulator whereas the *handleMessage()* method is called every simulation step and covers the permanent behaviour of the element.

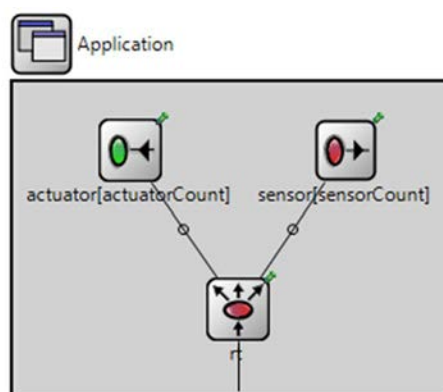
The *<project-name>.ini* and *<project-name>.ned* files participate in the definition of the network. The *<project-name>.ini* file contains the parameters and variables initialization values for a current simulation whereas in the *<project-name>.ned* file, the Network Descriptor (NED) language, allows the description of the topology of an OMNeT++ system. It has a programming syntax closed to an imperative language to define graph topologies of different shapes, such as, e.g. tree, ring, mesh, and their size. This is a real advantage of OMNeT++ over its direct competitor NS-3.

## 5.2 oneM2M deployment model

The simulator is organized into multiple modules either simple or compound with respect to the meta model defined in ETSI TS 103 840 [i.2]. The modules are organized into three categories: ApplicationLayer, ServiceLayer and InfrastructureLayer.

### Application Layer

The application layer is represented by a compound module named Application that hosts 3 types of submodules as showed in Figure 5.2-1.



**Figure 5.2-1: Application layer modules**

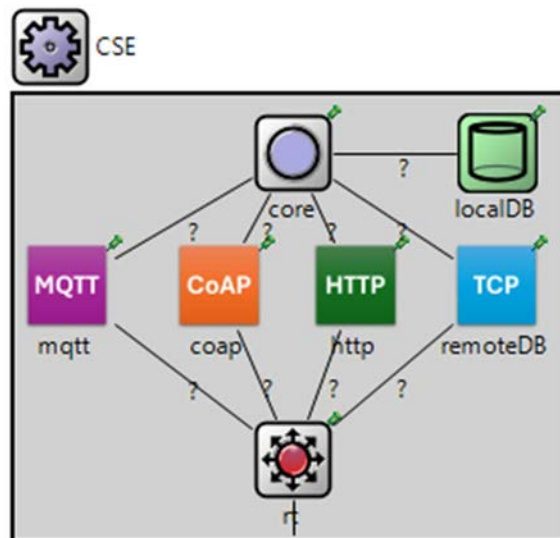
- **Actuator:Actuator** [0..N]: an Actuator is a simple module that receives messages from a remote CSE.
- **Sensor:Sensor** [0..N]: a Sensor is a simple module that sends messages to a remote CSE. One of the main parameters of this module is a JSON object representing a *CinGenerator* as defined in ETSI TS 103 840 [i.2]. The *CinGenerator* is a description of how Content Instances (CIN) messages are being generated by the sensor. The *CinGenerator* includes information on when a CIN is generated but also its size. Concerning the event generation, the simulator supports periodic generation, stochastic generation (uniform & exponential distribution), and generation following a time-series provided in an external file. Concerning the size of the message, the simulator supports constant size, stochastic size (uniform & normal distributions), and generation following a time-series provided in an external file.
- **rt:AppRouter** [1..1]: rt is a simple module that handles messages routing between sensors/actuators and the underlying communications services.

Both Sensor and Actuator modules have a parameter *IntializationProcedure* that contains the internal actions and messages sent to the remote CSE at start-up. Examples of these messages include the creation of oneM2M resources such as ApplicationEntity (AE), CoNtainTers (CNT), SUBscription (SUB) or initial ContentInstances (CIN).

Finally, an Application can host multiple Sensor and Actuator modules.

### Service Layer

The service layer is mainly represented by the compound module CSE as showed in Figure 5.2-2.



**Figure 5.2-2: Service layer modules**

Depending on its parameters, the CSE compound module hosts multiple simple modules that implement a feature of a oneM2M CSE. These modules are the following:

**core:Core** [1..1]: this mandatory module is responsible for handling CRUD operations received by a CSE through message passing. One of its main parameters is a JSON object that represents a oneM2M CSE Performance Descriptor (OCPD) as defined in ETSI TS 103 840 [i.2]. Based on this parameter, the CSE core can simulate the processing cost of a message and its associated operations in terms of computing (CPU) and memory (RAM) resources.

**mqtt:MQTTBinding** [0..1]: this optional module is responsible for managing the MQTT protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).

**coap:COAPBinding** [0..1]: this optional module is responsible for managing the CoAP protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).

**http:HTTPBinding** [0..1]: this optional module is responsible for managing the HTTP protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).

**remoteDB:TCPBinding** [0..1]: this optional module represents a data persistence service that is available in a remote host, and that needs a TCP communication. It is suitable for representing CSEs that use remote databases.

**localDB: DataStorage** [0..1]: this optional module represents a data persistence service that is embedded within the CSE. It is suitable for representing CSEs that use embedded databases (either file-based or in-memory database).

**rt:CSERouter** [1..1]: this is a simple module that handles messages routing between different components of the CSE and the underlying communication services.

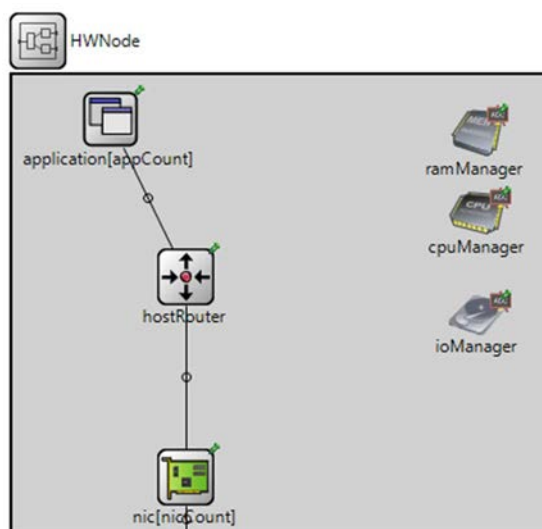
NOTE 1: A CSE has at least one protocol binding (HTTP, CoAP or MQTT) [i.6], [i.7] and [i.8].

NOTE 2: A CSE has one persistence service, either local or remote.

### Infrastructure Layer

The infrastructure layer is represented by modules related to hosting nodes (IoT, CSE, Generic Server) and to networking.

First, all hosting nodes derive from one generic node: the HWNode one. This compound module hosts the following modules, as showed in Figure 5.2-3.



**Figure 5.2-3: Infrastructure layer modules**

**application:Application** [0..N]: this compound module represents the IoT applications (cf. Application Layer) that run on the hosting node.

**hostRouter:HostRouter** [1..1]: this is a simple module that handles messages routing between different applications/services running on the hosting nodes and the underlying communication services (NICs).

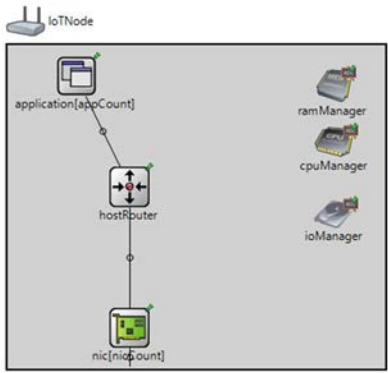
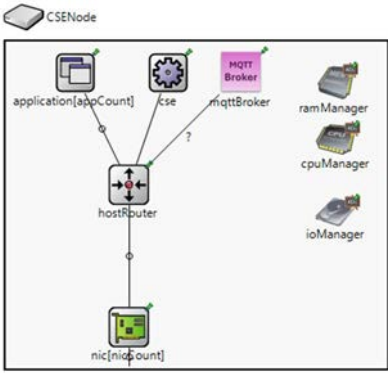
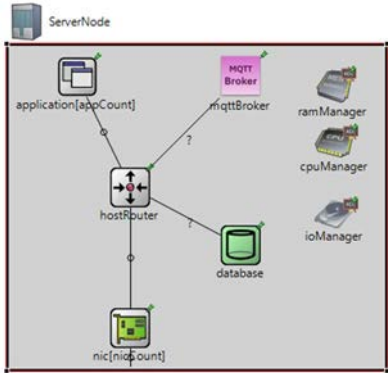
**ramManager:RAMManager** [1..1]: this is a simple module that tracks the overall usage of the processing power of the hosting node (i.e. CPU).

**cpuManager:CPUManager** [1..1]: this is a simple module that tracks the overall usage of the memory of the hosting node (i.e. RAM).



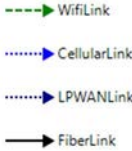
**ioManager:IOManager** [1..1]: this is a simple module that tracks the overall usage of the disk usage of the hosting node.

**nic:NIC** [1..N]: this simple module represents a networking interface available on the hosting node.

Based on this generic hosting node, three specific nodes are defined:

<b>IoTNode</b>	
<p>The IoTNode is a hosting node that run only IoT applications. An IoT application manages sensors and/or actuators.</p>	 <p>The IoTNode architecture diagram shows a central hostRouter connected to an application[appCount] above and a nic[nicCount] below. To the right of the hostRouter are three resource manager icons: ramManager, cpuManager, and ioManager.</p>
<b>CSENode</b>	
<p>The CSENode extends the generic HWNode and hosts one CSE. It can also host an MQTT Broker alongside the CSE. The connection between the hostRouter and the CSE represent the Mca/Mcc interfaces depending on whether the message arrives from an application or another CSE.</p>	 <p>The CSENode architecture diagram shows a central hostRouter connected to an application[appCount] above and a nic[nicCount] below. To the right of the hostRouter are three resource manager icons: ramManager, cpuManager, and ioManager. Above the hostRouter, there is a CSE and an MQTT Broker, both connected to the hostRouter with question marks indicating bidirectional communication.</p>
<b>ServerNode</b>	
<p>The ServerNode extends the HWNode by hosting an MQTT Broker and/or a database. This node is suitable for representing nodes acting as remote databases for a CSE, an independent MQTT Broker, or any application logic such as a monitoring application.</p>	 <p>The ServerNode architecture diagram shows a central hostRouter connected to an application[appCount] above and a nic[nicCount] below. To the right of the hostRouter are three resource manager icons: ramManager, cpuManager, and ioManager. Above the hostRouter, there is an MQTT Broker and a database, both connected to the hostRouter with question marks indicating bidirectional communication.</p>

The networking aspect is represented by two simple modules and four specific channels:

<p><b>NetworkElement:</b> This simple module represents a network router with basic IPv4 routing logic. It can be connected to any hosting node using a specific network channel.</p>	 <p>package oneM2M.Solution.InfrastructureLayer.Networking NetworkElement</p>
<p><b>Internet:</b> This is a simple module that is similar to the NetworkElement module since it acts as second hop router and connects all the network elements present in the IoT solution. It could be seen as the core network of telecom operators.</p>	 <p>package oneM2M.Solution.InfrastructureLayer.Networking Internet</p>
<p>Four channels are defined in the simulator. These channels represent the common communication links found in IoT solutions. Each channel defines its own data rate, latency, Bit Error Rate (BER) and Packet Error Rate (PER).</p>	 <p>package oneM2M.Solution.InfrastructureLayer.Networking WifiLink CellularLink LPWANLink FiberLink</p>

## 5.3 Mapping on OMNeT++ library and the NED topology

### 5.3.1 Mapping of Meta Model on OMNeT++

In order to start the simulation of an IoT solution that follows the Meta Model defined in ETSI TS 103 840 [i.2], two main files are provided in the simulator:

- First, the physical topology (Infrastructure Layer) of the IoT solution is defined in a file compliant with the syntax and semantics of OMNeT++'s description language (NED). This file describes the topology such a graph where vertices are one of the following: IoTNode, CSENode, ServerNode, NetworkElement, and Internet. These vertices are connected using one of the four links defined in the simulator: WiFiLink, CellularLink, LoRaLink, and FiberLink. The topology that can be seen as a tree with 3 levels:
  - Level 0 (root element): Internet node.
  - Level 1: nodes of type NetworkElement.
  - Level 2: nodes of type IoTNode, CSENode, or ServerNode.

A second file is the omnetpp.ini that contains values for the relevant parameters of the modules present in the topology file. All these parameters are of basic types such as String, IPAddress, Integer. Two specific parameters use structured data that can also be references to external files. These parameters are the following:

- **Event Generator:**
  - NED Module:
    - `oneM2M.Solution.ApplicationLayer.Sensor`
  - Usage (in INI file):
    - `<Solution>.<IoTNode>.application[appIndex].sensor[sensorIndex].cinGenerator`
  - Description: the ContentInstance generator parameter is a structured object using JSON format that implements a part of the oneM2M Application Scenario Descriptor (OASD) as defined in ETSI TS 103 840 [i.2]. It contains information about how a sensor generates oneM2M messages (i.e. CREATE operation of ContentInstance resource on the remote CSE) in terms of time instant and message data size. The schema for this object is given in the gitlab repository [i.11].



- **Performance Descriptor:**

- NED Modules:
  - `oneM2M.Solution.ServiceLayer.CSE`
  - `oneM2M.Solution.ServiceLayer.Components.Persistence.DataStorage`
- Usage (in INI file):
  - `<Solution>.<CSENode>.cse.performanceDescriptor`
  - `<Solution>.<CSENode>.cse.localDB.performanceDescriptor`
  - `<Solution>.<ServerNode>.database.performanceDescriptor`
- Description: the performance descriptor parameter is a structured object using JSON format that implements a part of the oneM2M CSE Performance Descriptor (OCPD) defined in [i.2]. Since this object contains information about system resources usage in terms of processing, memory, and disk usages for each CRUD operation per resource type, this parameter is provided for both the CSE module and the `DataStorage` module (embedded within the `CSENode` or hosted in a separate `ServerNode`). The schema for this object is given in the gitlab repository [i.11].

NOTE: These JSON parameters can be supplied inline in the INI file or in a separate file based on a prefix:

- Inline use in the INI file: `parameter = "data://<JSON_OBJECT>"`.
- External reference from the INI file: `parameter = "file://<FILE_PATH>"`.

### 5.3.2 Mapping of the probes in the simulator

Measurement probes are integrated in the source code of the simulator to build KPIs and evaluate the performance of the IoT system based on multiple simulation runs. These indicators include measures such as runtime, memory utilization, volume of data transferred, as well as specific metrics for each CRUD operations in the context of oneM2M resource creation on oneM2M objects. The associated KPIs are defined and described in clause 5 of ETSI TS 103 840 [i.2].

To operate, these probes are parameterized:

- First, by defining values to be integrated into the simulator such as importing the profiler results: for example, the size of the data produced by the sensors, the CPU delay to create a resource in a CSE, or the size of the ram used for this creation. This information is parameterized in the `.ini` file.
- A second part is to instantiate the probe with a variable that should evolve during the simulation run, for example, the total size of data produced, the total CPU duration for creating resources on a CSE. These variables are declared and managed in the C++ files of OMNeT++ elements.
- A third part is the declaration of the data format to be produced for the associated results. It can be either in scalar mode or more complex mode. This declaration of result format is specified in the `.ini` file.
- Last, these probes thus formed, register metrics values during simulations traces and aggregated and consolidated the results ranged as minimum, average and maximum values, or variance. These results can be linked to several characteristics of the system, i.e.: time computing, memory consumption, communication rate and data volume, just to mention a few.

## 5.4 Installation and configuration

The ETSI lab repository [i.11] contains the source codes of the two tools developed during the project. These tools: the *oneM2M-stack-profiler* and the *OMNeT-oneM2M-simulator* can be downloaded and experimented. The current clause explains how to use and experiment the performance evaluation of a oneM2M IoT system with the OMNeT++ simulator.

In a previous paragraph of clause 5, the underlying concepts of the simulator have been presented. The current clause explains how to play with the simulator and the implemented use case.

The tool repository for the *oneM2M simulator* contains two folders:

- *simulation\_results* is a well-named folder that contains an extract of the different simulation runs.
- *src* contains the source code of the OMNeT++ simulator and the different topologies (.ned files).
- The development has been conducted with OMNeT++ V6.0.3.

Here is a fast installation procedure to run the simulator:

- 1) Download and install:
  - OMNeT++ on [i.10].
  - the code contained in the *src* directory from the GitLab repository [i.11].
- 2) Start the IDE OMNeT++:
  - see the OMNeT++ installation guide [i.12].
- 3) Create an OMNeT++ project:
  - Choose "Empty Project with src and simulations folders".
  - Delete a package.ned file that has been generated in the project.
  - In the src folder -> Import code file system by selecting the local src folder that contains the code.
- 4) Build the project:
  - Click right on the package -> "Build Project".
- 5) Run a simulation:
  - Click right on the package -> "Run As" -> "Run Simulation".
  - Choose the *src/omnetpp.ini* as configuration file.
  - The user is now able to run multiple simulations by choosing different topologies namely (...).

## 6 Frameworks at work

### 6.1 Mapping Scenario Description

#### 6.1.1 Running example: A traffic light system

To simulate the Traffic Light use case presented in ETSI TS 103 840 [i.2], the first step is to translate the corresponding oneM2M Solution Deployment Descriptor (OSDD) into a NED topology file in OMNeT++. Based on the infrastructure view of an IoT solution, as showed in Figure 6.1.1-1 (left) this translation is represented in OMNeT++ as per Figure 6.1.1-1 (right):

- The solution nodes are represented by their corresponding NED module from the simulation library: IoTNode, CSENode or ServerNode.
- Each AccessNetwork is represented by a NetworkElement NED module. Connections between the relevant nodes and this NetworkElement use the corresponding link type. For example, the AccessNetwork is Wi-Fi®, the links between the nodes and the corresponding NetworkElement will use WiFiLink channel.
- Finally, the Internet entity in the model is represented by the specific NED module: Internet. All links between the network elements and the Internet node are assumed of FiberLink type.

NOTE: This translation phase can benefit from automation to address large instances of IoT solutions that is out of the scope of ETSI TTF T019.

The second step is to supply an OMNeT++ simulation parameters file a.k.a. INI file that contains the values of all the relevant parameters of the involved NED modules.

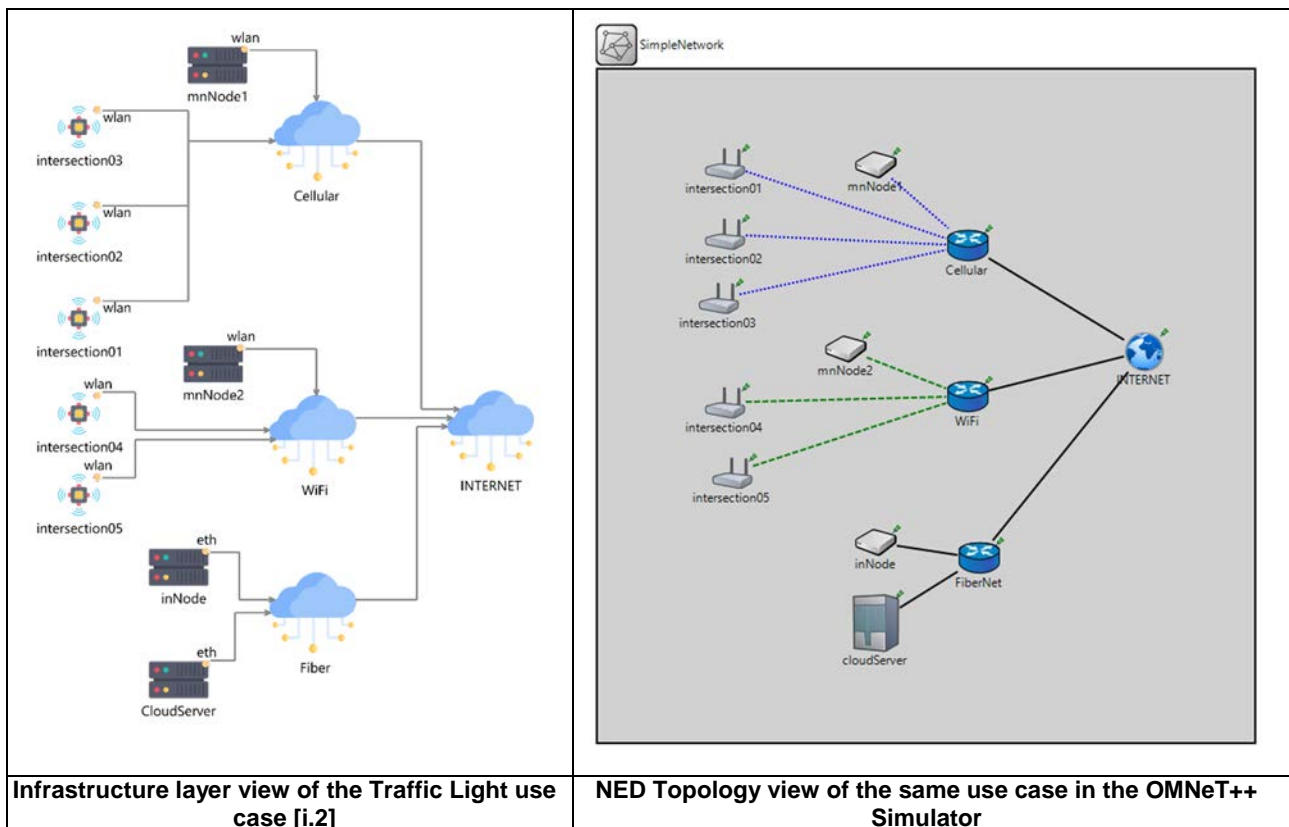


Figure 6.1.1-1: Infrastructure view and their associated OMNeT++ view

## 6.1.2 OMNeT++ files

The corresponding NED topology file for this example is as follows:

```

package onem2msimulator.simulations;
import oneM2M.Solution.InfrastructureLayer.Networking.CellularLink;
import oneM2M.Solution.InfrastructureLayer.Networking.FiberLink;
import oneM2M.Solution.InfrastructureLayer.Networking.WifiLink;
import oneM2M.Solution.InfrastructureLayer.Networking.NetworkElement;
import oneM2M.Solution.InfrastructureLayer.Networking.Internet;
import oneM2M.Solution.InfrastructureLayer.Nodes.CSENode;
import oneM2M.Solution.InfrastructureLayer.Nodes.IoTNode;
import oneM2M.Solution.InfrastructureLayer.Nodes.ServerNode;

network SimpleNetwork
{
  parameters:
    // name: Traffic Light Use Case
    // description: A traffic Light scenario with 3 CSEs, 5 IoTNodes connected to 3 different access networks
    // authors: TTF T019 Experts
    // version: 1.0

  submodules:
    cloudServer: ServerNode;
    mnNode1: CSENode;
    mnNode2: CSENode;
    inNode: CSENode;
    FiberNet: NetworkElement;
    Cellular: NetworkElement;
    WiFi: NetworkElement;
    INTERNET: Internet;
    intersection01: IoTNode;
    intersection02: IoTNode;
    intersection03: IoTNode;

```

```

intersection04: IoTNode;
intersection05: IoTNode;
connections:
mnNode1.toNetwork[0] <--> CellularLink <--> Cellular.link++;
intersection03.toNetwork[0] <--> CellularLink <--> Cellular.link++;
intersection02.toNetwork[0] <--> CellularLink <--> Cellular.link++;
intersection01.toNetwork[0] <--> CellularLink <--> Cellular.link++;

mnNode2.toNetwork[0] <--> WifiLink <--> WiFi.link++;
intersection04.toNetwork[0] <--> WifiLink <--> WiFi.link++;
intersection05.toNetwork[0] <--> WifiLink <--> WiFi.link++;

inNode.toNetwork[0] <--> FiberLink <--> FiberNet.link++;
cloudServer.toNetwork[0] <--> FiberLink <--> FiberNet.link++;

WiFi.internetLink <--> FiberLink <--> INTERNET.link++;
Cellular.internetLink <--> FiberLink <--> INTERNET.link++;
FiberNet.internetLink <--> FiberLink <--> INTERNET.link++;
}

```

The corresponding INI file for this example is as follows:

#### [General]

```
network = onem2msimulator.simulations.SimpleNetwork
```

```
SimpleNetwork.intersection*.appCount = 1
SimpleNetwork.intersection*.application[0].name = "app1"
SimpleNetwork.intersection*.application[0].sensorCount = 1
SimpleNetwork.intersection*.application[0].sensor[0].name = "intersectionState"
SimpleNetwork.intersection*.application[0].actuatorCount = 1
SimpleNetwork.intersection*.application[0].actuator[0].name = "threeLights"
```

```
SimpleNetwork.mnNode1.name = "mn1-cse"
SimpleNetwork.mnNode2.name = "mn2-cse"
SimpleNetwork.inNode.name = "in-cse"
```

```
SimpleNetwork.cloudServer.nic[0].name = "eth0"
SimpleNetwork.inNode.nic[0].name = "eth0"
SimpleNetwork.mnNode*.nic[0].name = "wlan0"
SimpleNetwork.intersection*.nic[0].name = "wlan0"
```

```
SimpleNetwork.FiberNet.subnetAddress = "10.0.1.0/24"
SimpleNetwork.cloudServer.nic[0].networkAddress = "10.0.1.10"
SimpleNetwork.inNode.nic[0].networkAddress = "10.0.1.11"
```

```
SimpleNetwork.Cellular.subnetAddress = "10.0.2.0/24"
SimpleNetwork.mnNode1.nic[0].networkAddress = "10.0.2.10"
SimpleNetwork.intersection01.nic[0].networkAddress = "10.0.2.11"
SimpleNetwork.intersection02.nic[0].networkAddress = "10.0.2.12"
SimpleNetwork.intersection03.nic[0].networkAddress = "10.0.2.13"
```

```
SimpleNetwork.WiFi.subnetAddress = "10.0.3.0/24"
SimpleNetwork.mnNode2.nic[0].networkAddress = "10.0.3.10"
SimpleNetwork.intersection04.nic[0].networkAddress = "10.0.3.11"
SimpleNetwork.intersection05.nic[0].networkAddress = "10.0.3.12"
```

```
SimpleNetwork.intersection01.application[0].remoteCSE = "mn1-cse"
SimpleNetwork.intersection02.application[0].remoteCSE = "mn1-cse"
SimpleNetwork.intersection03.application[0].remoteCSE = "mn1-cse"
SimpleNetwork.intersection04.application[0].remoteCSE = "mn2-cse"
SimpleNetwork.intersection05.application[0].remoteCSE = "mn2-cse"
SimpleNetwork.mnNode1.cse.remoteCSE = "in-cse"
SimpleNetwork.mnNode2.cse.remoteCSE = "in-cse"
```

```
SimpleNetwork.intersection01.application[0].remoteCSEAddress = "10.0.2.10"
SimpleNetwork.intersection02.application[0].remoteCSEAddress = "10.0.2.10"
SimpleNetwork.intersection03.application[0].remoteCSEAddress = "10.0.2.10"
SimpleNetwork.intersection04.application[0].remoteCSEAddress = "10.0.3.10"
SimpleNetwork.intersection05.application[0].remoteCSEAddress = "10.0.3.10"
SimpleNetwork.mnNode1.cse.remoteCSEAddress = "10.0.1.11"
SimpleNetwork.mnNode2.cse.remoteCSEAddress = "10.0.1.11"
```

```
SimpleNetwork.intersection01.application[0].sensor[0].cinGenerator = "data://{ \"type\": \"SimulatedEvent\",
\"eventDistribution\": { \"type\": \"Constant\", \"constant\": 13.0 }, \"dataSizeDistribution\": {
\"type\": \"Constant\", \"constant\": 100 }}"
```

```

SimpleNetwork.intersection02.application[0].sensor[0].cinGenerator = "file://sample_cin_generator_v1.json"
SimpleNetwork.intersection03.application[0].sensor[0].cinGenerator = "file://sample_cin_generator_v1.json"
SimpleNetwork.intersection04.application[0].sensor[0].cinGenerator = "file://sample_cin_generator_v1.json"
SimpleNetwork.intersection05.application[0].sensor[0].cinGenerator = "file://sample_cin_generator_v1.json"

SimpleNetwork.mnNode1.cse.performanceDescriptor = "file://acme_performances_v1.json"
SimpleNetwork.mnNode2.cse.performanceDescriptor = "file://acme_performances_v1.json"
SimpleNetwork.inNode.cse.performanceDescriptor = "file://acme_performances_v1.json"

SimpleNetwork.mnNode1.cse.localDB.performanceDescriptor = "file://sample_memory_db_performances_v1.json"
SimpleNetwork.mnNode2.cse.localDB.performanceDescriptor = "file://sample_memory_db_performances_v1.json"

SimpleNetwork.inNode.hasLocalDB = false
SimpleNetwork.inNode.cse.dbServerAddress = "10.0.1.10"
SimpleNetwork.cloudServer.hasDataBase = true
SimpleNetwork.cloudServer.database.performanceDescriptor = "file://sample_server_db_performances_v1.json"

```

### 6.1.3 Synthetizing KPIs

Simulation results are recorded using OMNeT++ built-in support. This support is mainly achieved through two SDK tools:

- **output vectors (cOutVector):** output vectors are time series data, recorded from simple modules or channels. They can be used to record evolving data over time such as queue length, processing time, end-to-end delays, etc. Every output vector is identified by a name. Recording is done through the method:
  - cOutVector::recordWithTimestamp(timestamp, value).
- **output scalars (cStdDev):** output scalars are summary results. They are computed during the simulation and written out at the end. They can be single integer/real values, or a statistical summary composed of multiples fields such as mean, standard deviation, sum, minimum, maximum, etc. They can also be used to store histogram data. As for output vectors, output scalars are identified by a name. Recording data is done through the method:
  - cStdDev::collect(value).

Output vectors and scalars are inserted though out the simulation modules, in particular:

- Memory usage in every HWNode and its subclasses (IoTNode, CSENode, ServerNode);
- Processing time in every CSENode and especially in the CSE Core module;
- Processing time for every storage module (local or remote);
- End-to-End delay in every Sensor module (time span between the instant the oneM2M request is issued by the sensor and the instant the oneM2M response is received by the sensor);
- etc.

At the end of the simulation, all simulations results are automatically stored in separate files (\*.vec and \*.sca) that can be further processed by OMNeT++ for visualization or data analysis.

## 6.2 Performance evaluation analysis

Simulation can be run both in graphical mode and in silent mode (no GUI). In graphical mode, different views are proposed: timeline, scheduled events, topology, etc. as shown in Figure 6.2-1.

The screenshot displays the OMNeT++ simulation interface in graphical mode. The main window shows a network diagram titled "SimpleNetwork" with various components: cloudServer, mnNode1, mnNode2, inNode, intersection01-05, Cellular, WiFi, FiberNet, and INTERNET. The diagram illustrates connections between these elements, such as Cellular and WiFi nodes connected to the INTERNET, and various intersection nodes connected to the Cellular and WiFi nodes.

The left sidebar shows a tree view of the simulation components, including a list of scheduled events (simulation.scheduled-events) and a list of events (events[132]). The bottom status bar indicates "Msg stats: 132 scheduled / 1960 existing / 4994 created".

The bottom right corner shows a list of events (Event #12604 to #12623) with their corresponding timestamps and descriptions, such as "Event #12604 t=299 SimpleNetwork.intersection04.hostRouter (InternalRouter, id=103) on CREATE CIN (NetworkPacket, id=142602)".

Figure 6.2-1: OMNeT++simulation run in graphical mode

Any output vector declared in the simulation is listed under the parent module. Output vectors can be visualized in real-time during the simulation (one new window per output vector). Results, using the GUI OMNeT++ interface, are displayed in Figures 6.2-2, 6.2-3 and 6.2-4.



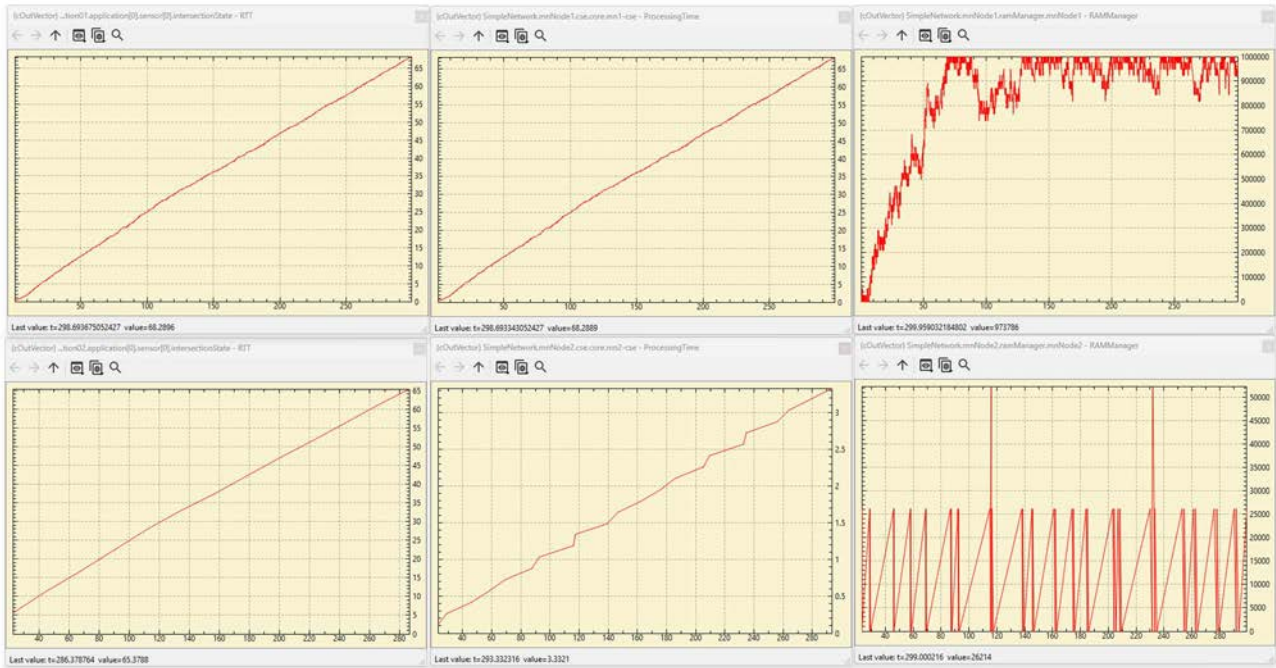


Figure 6.2-2: real-time visualization of simulation results through output vectors.

At the end of the simulation, OMNeT++ offers a complete set of tools for analysing the simulation results. A list of output vectors and scalars are summarized. The user can plot the simulation results individually or combined. Meta analysis such as difference, division, linear trend, moving average, etc. can be additionally computed.

Browse Data

Here you can see all data that come from the files specified in the Inputs page.

All (482 / 482) Parameters (466 / 466) Scalars (0 / 0) Histograms (0 / 0) Vectors (16 / 16)

experiment filter measurement filter replication filter module filter result name filter

Experiment	Measure...	Replication	Module	Name	Count	Mean	StdDev	Variance
General	#0	#0	SimpleNetwork.cloudServer.ramManager	cloudServer - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.inNode.ramManager	inNode - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection01.ramManager	intersection01 - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection02.ramManager	intersection02 - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection03.ramManager	intersection03 - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection04.ramManager	intersection04 - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection05.ramManager	intersection05 - RAMManager	1	0	-	-
General	#0	#0	SimpleNetwork.intersection05.application[0].sensor[0]	intersectionState - RTT	10	1.800532	1.046555	1.095278
General	#0	#0	SimpleNetwork.intersection04.application[0].sensor[0]	intersectionState - RTT	12	1.660099	0.991685	0.983439
General	#0	#0	SimpleNetwork.intersection03.application[0].sensor[0]	intersectionState - RTT	4	42.255139	23.477314	551.184384
General	#0	#0	SimpleNetwork.intersection02.application[0].sensor[0]	intersectionState - RTT	12	36.235656	19.947061	397.885233
General	#0	#0	SimpleNetwork.intersection01.application[0].sensor[0]	intersectionState - RTT	427	33.958771	19.667801	386.822395
General	#0	#0	SimpleNetwork.mnNode1.cse.core	mn1-cse - ProcessingTime	443	34.094695	19.677599	387.207908
General	#0	#0	SimpleNetwork.mnNode2.cse.core	mn2-cse - ProcessingTime	22	1.723500	0.994817	0.989662
General	#0	#0	SimpleNetwork.mnNode1.ramManager	mnNode1 - RAMManager	2027	819.272622595	246.761389522	60.891183.358.794790
General	#0	#0	SimpleNetwork.mnNode2.ramManager	mnNode2 - RAMManager	90	14.272066667	14.256061736	203.235.296.220225

Figure 6.2-3: OMNeT++ simulation results browser

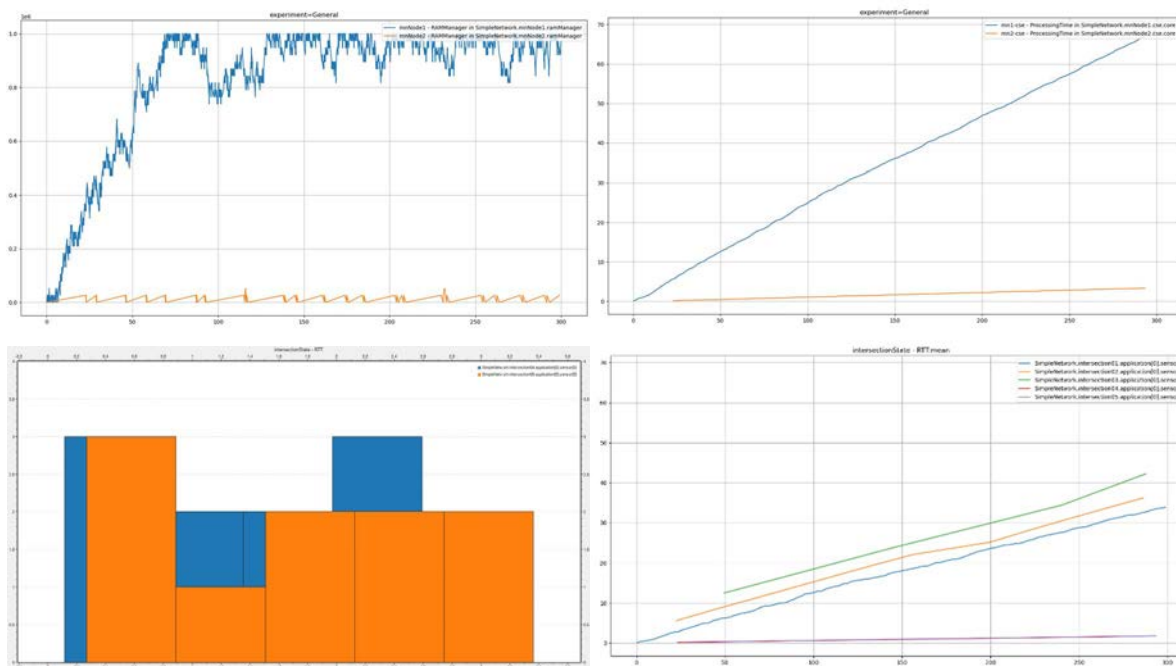


Figure 6.2-4: OMNeT++ results viewer

## 7 Conclusions

The present document details the two tools developed in ETSI TTF T019, namely a profiler, written in Python and a simulation library, written in the OMNeT++ simulator. The first one allows to profile the use of hardware resources by a subset of resources of a CSE of the oneM2M standard. The profiler can run different oneM2M stacks and observe their hardware usage based on a set of oneM2M resources calls. The second one allows to simulate a complete deployment of an oneM2M architecture, extrapolating and scaling up the information provided by the profiler and carrying out a complex and large scale IoT application.

Clause 4 describes how the profiler works and the measurements it provides.

Clause 5 presents a complete mapping of oneM2M meta model into OMNeT++ objects. The modules and their relationship are explained in relation to ETSI TS 103 840 [i.2] describing the used meta-models.

Clause 6 shows the use of the simulator in a simple scenario. ETSI TR 103 842 [i.3] will complete the usage based on the scenarios defined in ETSI TR 103 839 [i.1].



---

## Annex A: Source code

See <https://labs.etsi.org/rep/iot/smartm2m-onem2m-performance-evaluation/profiler>.

See <https://labs.etsi.org/rep/iot/smartm2m-onem2m-performance-evaluation/onem2m-simulator>.

---

## History

<b>Document history</b>		
V1.1.1	July 2024	Publication